Frequency Counter on FPGA

Introduction to digital Low-Level Radio Frequency Controls in Accelerators

Lab 4

Qiang Du

US Particle Accelerator School
Janurary 23 – 27, 2023

Contents

1 Introduction

This is a demo about a frequency meter, which is extremely important for LLRF applications.

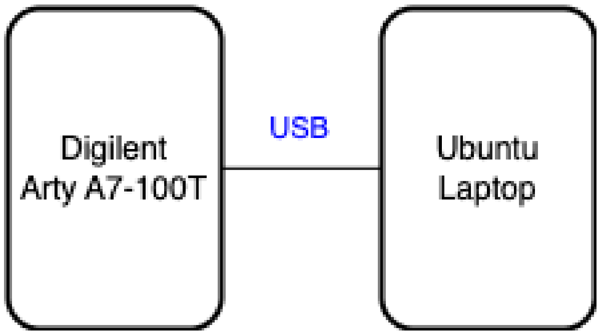
The core design is from Larry Doolittle as in `freq_count.v`, with test bench.

The rest is to make a multiple channels frontend (`freq_multi_count_fe.v`), and code for human interface, including a binary-coded decimal decoder (`b2decimal.v`), a `printf()`-like ASCII character translator, and a simple UART driver (`simpleuart.v`), all wrapped together within `freq_demo.v`, which was instantiated by the top level `freq_demo_top.v`, where the *unknown* clock is generated by a typical Xilinx PLL primitive (`MMCM`) at the expected frequency of 100 MHz.

The hardware is the Digilent Arty A7 FPGA, where the reference clock is also at 100MHz.

The goal of this lab is to get familiar with the concept of clock domain crossing, gray code, UART, etc.

1.1 Hardware setup



2 Exercise

2.1 RTL simulation of `freq_count.v`

With the command of:

```
make freq_count.vcd
```

Expect terminal output:

```
vvp -N freq_count_tb +vcd
VCD info: dumpfile freq_count.vcd opened for output.
time:    200 ns, measured freq:   0.00 MHz, error =  166.7 MHz
time:    600 ns, measured freq:   0.00 MHz, error =  166.7 MHz
time:   1000 ns, measured freq:   0.00 MHz, error =  166.7 MHz
time:   1400 ns, measured freq: 154.69 MHz, error =   12.0 MHz
time:   1800 ns, measured freq: 154.69 MHz, error =   12.0 MHz
time:   2200 ns, measured freq: 154.69 MHz, error =   12.0 MHz
```

```
time:   2600 ns, measured freq: 167.19 MHz, error =   -0.5 MHz
time:   3000 ns, measured freq: 167.19 MHz, error =   -0.5 MHz
time:   3400 ns, measured freq: 167.19 MHz, error =   -0.5 MHz
time:   3800 ns, measured freq: 167.19 MHz, error =   -0.5 MHz
time:   4200 ns, measured freq: 166.41 MHz, error =    0.3 MHz
PASS
```

Review the testbench `freq_count_tb.v`, vary the unknown clock period by line:

```
parameter integer FCLK_PERIOD = 6;      // unknown clock period in ns
```

Observe the result.

2.2 Understand the concept of `freq_count.v`

Review the source code of `freq_count.v`. Understand the concept of clock domain crossing using gray code, and the need for binary to gray code conversion twice, in both clock domains: unknown frequency domain and the reference domain. The heart of it is the measurement of the difference of the binary representation of a 4-bit counter after the domain crossings at every clock cycle. The difference is then accumulated (or averaged) within the duration defined by its data width (parameter `refcnt_width`), and the result is strobed out once the accumulation cycle is done. Therefore the result is the fractional ratio between the unknown and reference frequencies with `refcnt_width` bit resolution.

2.3 Synthesize bitstream file for Arty A7

Now we can put everything together and synthesize the bitstream file for measuring the 100MHz clock on Arty A7 board.

```
make freq_demo_top.bit
```

2.4 Test on hardware

Program the bitfile by:

```
make config_arty_freq
```

Open a UART terminal (baud rate 9600,8,N,1):

```
pyserial-miniterm /dev/ttyUSB0
```

where `ttyUSB0` assumes there is only one board connected.

Expected results:

```
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---

     Channel 1:  000.00000 MHz
     Channel 2:  000.00000 MHz
     Channel 3:  000.00000 MHz
Channel 0:  099.99999 MHz
```

2.5 Change Frequency

Change the *unkown* frequency by reconfiguring the `MMCM` module using the division factor (`DIV0` and `DIV1`) at `freq_demo_top.v`:

```
xilinx7_clocks #(
    .DIFF_CLKIN     ("FALSE"),  // Single ended
    .CLKIN_PERIOD   (10.0),     // 100 MHz
    .MULT           (10),       // 1000 MHz
    .DIV0           (8),        // 125 MHz
    .DIV1           (10)        // 100 MHz
) xilinx7_clocks_i (
    .sysclk_p   (CLK100MHZ),
    .sysclk_n   (1'b0),
    .sysclk_buf (sysclk_buf),
    .reset      (reset),
    .clk_out0   (clk125),
    .clk_out1   (clk100),
    .locked     (locked)
);
```

Re-synthesize the bitfile and test.